Component Verification and Certification in NASA Missions

Dimitra Giannakopoulou
RIACS

John Penix
Computational Sciences Division

Automated Software Engineering Group
NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
{dimitra, jpenix}@ptolemy.arc.nasa.gov

Abstract

Software development for NASA missions is a particularly challenging task. Missions are extremely ambitious scientifically, have very strict time frames, and must be accomplished with a maximum degree of reliability. Verification technologies must therefore be pushed far beyond their current capabilities. Moreover, reuse and adaptation of software architectures and components must be incorporated in software development within and across missions. This paper discusses NASA applications that we are currently investigating from these perspectives.

1. Introduction

The challenges that NASA faces in software development are untypical of most organizations. Preparation for a single NASA mission may incur billions of dollars, which makes loss of a mission a very serious matter. Launch windows are often inflexible. For example Mars missions may only be launched every 26 months, for planets to be in specific favorable positions. Failing to prepare such a mission on time is equivalent to loss of mission. There is therefore, often, a very strict "time to market". These reasons make the organization particularly sensitive to issues of reliability, and conservative in accepting new technologies that cannot be verified adequately.

As extremely ambitious projects are aimed for by NASA, software plays a greater role in its missions. For example, compared to the 32,000 lines of code required for the Cassini space craft, the International Space Station contains 2 million lines of code operating on 52 computers. Such software is typically developed by distributed interdisciplinary teams of contractors and NASA engineers. Information and knowledge about software process and products must be exchanged across organizational boundaries inside and outside of NASA. As a result, component integration is a critical issue that has to be addressed and verified at all phases of software development.

The current trend within NASA is toward more-frequent lower-cost missions. Even when missions have different objectives, there are a lot of common decisions made during mission planning and design. Therefore, to achieve the required quality within limited mission budgets and time frames, NASA must develop software architectures that can be reused across missions and components that can be adapted to specific mission requirements.

In what follows, we outline a number of NASA application areas in which we conduct research related to component architectures, integration, and verification.

2. NASA Applications

This section provides an overview of challenging application areas we are investigating, of research issues they involve, and of targeted technologies.

2.1 Autonomous Architectures

Autonomous software systems pose new challenges to software verification. They involve complex concurrent behaviors for reacting to external stimuli and possibly unpredictable environments without human intervention. Such an example is software for Rovers for Mars missions. In the past, a Rover would receive from ground plans corresponding to fully deterministic/inflexible executions of the Rover. In future missions, these plans will allow an increasing degree of flexibility. The Rover will be able to make behavioral decisions based on environmental or timing conditions. This results in a huge number of possible executions that would need to be thoroughly verified to avoid loss of time, resources, or even loss of mission.

Extensive verification and validation are therefore pre-requisites for the deployment of missions that involve autonomous systems. For such complex systems, however, traditional testing techniques have proven insufficient in providing the desired degree of correctness guarantees. For example, despite extensive testing, the Remote Agent, support the verification of IMA architectures and applications running on IMA. Initial investigations with DEOS, a microkernel-based real-time operating system used in Honeywell's Primus Epic avionics product line, have shown that model-checking can be effective in verifying critical properties of these architectures [9]. However, more research is needed to scale the technology to full-size IMA systems and to address independent certification of IMA applications [10].

The cost-saving promises of independent certification and reuse-based certification are attractive elements of the IMA approach. However, it is well known that, except in the most controlled environments, software reuse quickly becomes an ad-hoc process with unpredictable results. To address this issue, we are developing component-based application generation technology, which produces a certification argument during the generation process. This work is an extension of previous work that uses automated theorem proving technology to generate software from subroutine libraries along with explanations of how the software was derived [11, 12]. The approach we are investigating is to use formal specifications to represent both the generic behavior of the reusable components as well as adaptation and configuration techniques that are used to integrate the component into the generated application. It is our hypothesis that separating the specification of component function from interaction will provide a more effective framework for constructing certification arguments.

2.3 Component-Based Scientific Programming

Data analysis and visualization are major activities required for the support of NASA science missions. Numerous software toolkits and libraries are available to support the construction of custom data analysis and visualization applications. However, due to differences in data formats and system implementation styles, using these systems often requires space scientists to perform extensive custom programming that is time-consuming, expensive and not reusable between missions. In addition, current approaches to developing scientific software do not scale well. Scientific software is commonly developed from prototype systems and is evolved through experimentation. As the software is expanded and generalized, it becomes difficult to modify and maintain due to this ad-hoc development style. Because scientific problems are becoming larger and more complex, these software-engineering issues are unavoidable.

To address these problems, there has been a recent movement toward component-based software development for scientific programming. Increased reliability is the primary benefit of a component-based approach from the perspective of scientific applications, because unreliable software leads to unreliable scientific results [13]. In addition, it simplifies component integration and can address the scalability and maintainability problems of scientific applications.

To support the use of component-based software for web-based data analysis applications, we are integrating component-based application generation technology into a middleware infrastructure [14]. Using this infrastructure, scientists will provide a specification of three kinds of components: (1) the desired data source, (2) a data analysis filter, and (3) a visualization device. From this information, the desired application will be generated automatically. To provide this automated capability, the infrastructure contains an intelligent component integration system (iCIS) that automates component retrieval, adaptation, and integration using automated reasoning [15-17]. It takes the user's request, locates the corresponding components in a heterogeneous and distributed environment, and uses component interface specification and data archive meta-data to automatically generate any adapters required for component integration. Thus, the need for time-consuming and expensive custom programming will be reduced.

3. Discussion

The research projects that we have described in this paper are at very early stages. We are still trying to capture the complexity of the problems they involve, let alone solve those problems.

Modularity can pay off during all phases of system development. It is the most promising factor in scaling system verification; it can make systems more flexible, in that it facilitates replacing components or plugging in new ones. To achieve it is, however, far from straightforward. It requires careful design of a system as early as at the architecture stage.

One of the issues we need to address within this context is to provide support for checking systems or their components in isolation. This requires providing appropriate environments or contexts for them. A common factor in the applications that we have described is that the environments in which they operate are complex and unpredictable, so that it becomes very difficult to provide any type of constraints for their possible behaviors. Moreover, the complex relationships that exist between the components in these systems make it difficult to construct context/environment models that can be used to verify non-trivial properties compositionally.

An additional issue has to do with defining component interfaces and checking their compatibility, but for information that involves behavioral, rather than simply syntactic aspects. However, the type, degree of detail, and form that this information must take to support tractable verification of properties of practical interest is an open research issue.

There remains a lot of work to be done towards developing good software engineering principles in general. Given the particular nature and complexity of the problems we are faced with, we are first working toward domain-specific solutions as a more realistic goal. We anticipate that architectures for our target applications will gradually evolve to provide some verifiable guarantees about domain-specific requirements.

Ultimately, we would like to be able to advise engineers about how to design and structure their systems for verifiability. The need for a maximum degree of reliability in NASA missions would justify adapting the software and systems developed to the rules that the need for verifiability might provide. As the architectures and tools mature, it may be possible to use domain-specific application generation technology to support reuse of these verified architectures.

Acknowledgements

We thank the members of the Automated Software Engineering Group at NASA Ames and others involved in our research for their participation and collaboration on the research projects mentioned in this paper.

References

[1] Havelund, K., *et al.* "Formal Analysis of the Remote Agent Before and After Flight", in *Proc. of the 5th NASA Langley Formal Methods Workshop*. 2000.

[2] Giannakopoulou, D., Kramer, J., and Cheung, S.C., *Analysing the Behaviour of Distributed Systems using Tracta.* Journal of Automated Software Engineering, special issue on Automated Analysis of Software, Vol. 6(1), Kluwer Academic Publishers, January 1999: pp. 7-35.

[3] Henzinger, T.A., Qadeer, S., and Rajamani, S.K. "You assume, we guarantee: methodology and case studies", in *Proc. of the International Conference on Computer-Aided Verification (CAV'98).* June 28 - July 2, 1998, Vancouver, Canada. Springer, Lecture Notes in Computer Science (LNCS) 1427, pp. 440-451. A.J. Hu and M.Y. Vardi, Eds.

[4] McMillan, K.L., *A methodology for hardware verification using compositional model checking.* Science of Computer Programming, Vol. 37, Elsevier: pp. 279-309.

[5] Gamble, E. and Simmons, R., *The Impact of Autonomy Technology on Spacecraft Software Architecture: A Case Study.* IEEE Intelligent Systems, IEEE, September/October 1998.

[6] Dvorak, D., Rasmussen, R., Reeves, G., and Sacks, A. "Software Architecture Themes in JPL's Mission Data System", in *Proc. of the 2000 IEEE Aerospace Conference.* March 2000, Big Sky, Montana.

[7] Visser, W., Havelund, K., Brat, G., and Park, S. "Model Checking Programs", in *Proc. of the 15th IEEE International Conference on Automated Software Engineering (ASE'2000).* 11-15 September 2000, Grenoble, France. IEEE Computer Society, pp. 3-11. Y. Ledru, P. Alexander, and P. Flener, Eds.

[8] Havelund, K. "Using Runtime Analysis to Guide Model Checking of Java Programs", in *Proc. of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification.* August/September 2000, Stanford, CA. Springer, Lecture Notes in Computer Science 1885. K. Havelund, J. Penix, and W. Visser, Eds.

[9] Penix, J., *et al.* "Verification of Time Partitioning in the DEOS Scheduler Kernel", in *Proc. of the 22nd International Conference on Software Engineering (ICSE'2000).* 2000, Limeric, Ireland. ACM Press.

[10] Cofer, D., *et al.* "Using Model Checking for Verification of Partitioning Properties in Integrated Modular Avionics", in *Proc. of the 2000 IEEE Digital Avionics Systems Conference.* 2000.

[11] VanBaalen, J., Robinson, P., Lowry, M., and Pressburger, T. "Explaining Synthesized Software", in *Proc. of the 13th IEEE Conference on Automated Software Engineering.* October 13-16, 1998, Honolulu, Hawaii.

[12] Stickel, M., *et al.* "Deductive Composition of Astronomical Software from Subroutine Libraries", in *Proc. of the 12th Conference on Automated Deduction.* June 28-July 1, 1994, Nancy, France. Springer-Verlag, Lecture Notes in Computer Science 814. A. Bundy, Ed.

[13] Dubois, P.F., *Scientific Components are Coming.* IEEE Computer, March 1999.

[14] Penix, J., *et al.* "Automating Component Integration for Web-Based Data Analysis", in *Proc. of the 2000 IEEE Aerospace Conference.* March 2000, Big Sky, Montana.

[15] Penix, J. "Deductive Synthesis of Event-Based Software Architectures", in *Proc. of the 14th International Automated Software Engineering Conference.* October 1999, Cocoa Beach, Florida.

[16] Penix, J. and Alexander, P., *Efficient Specification-Based Component Retrieval.* Automated Software Engineering, Vol. 6, Kluwer Academic Publishers, April 1999: pp. 139-170.

[17] Fischer, B. and Whittle, J. "An Integration of Deductive Retrieval into Deductive Synthesis", in *Proc. of the 14th International Automated Software Engineering Conference.* October 1999, Cocoa Beach, Florida.